

Министерство образования и науки РФ  
ФГБОУ ВО ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Информационная безопасность систем и технологий»

ОТЧЕТ  
о лабораторной работе №6  
ШАБЛОНЫ В СИ++

Дисциплина: Языки программирования

Группа: 18ПИ1

Выполнил: Новиков Д.О.

Количество баллов:

Дата сдачи:

Принял: к.т.н., доцент Лупанов М.Ю.

Пенза 2019

## 1 Цель работы

1.1 Освоить создание и использование шаблонов функций и шаблонов классов в программах на языке Си++ .

## 2 Задание к лабораторной работе

2.1 Реализовать алгоритм сортировки (алгоритм выбрать самостоятельно) в виде шаблонной функции. Продемонстрировать работу шаблонной функции на массивах различных типов.

2.2 Реализовать класс `Rectangle` с атрибутами, хранящими высоту и ширину, и поддерживающий 40 сравнение по площади. Продемонстрировать сортировку массива объектов типа `Rectangle` шаблонной функцией, разработанной в предыдущем задании.

2.3 Реализовать шаблонный класс `DoubleBox` для хранения двух атрибутов разного типа. Тип атрибутов задать параметрами шаблона. Реализовать в классе конструктор по умолчанию, инициализирующий конструктор, а также методы `get` и `set`.

2.4 Задание повышенной сложности. Реализовать шаблонный класс `Array` для хранения массива произвольного типа и размера (без использования динамической памяти). Тип элементов и размер массива задать параметрами шаблона. Реализовать в классе следующие конструкторы: – конструктор по умолчанию, без параметров; – конструктор, позволяющий инициализировать весь внутренний массив одинаковыми значениями, имеющий один параметр — инициализирующее значение; – конструктор, позволяющий инициализировать внутренний массив значениями из внешнего массива, имеющий два параметра — указатель на внешний массив и размер внешнего массива. Реализовать в классе `Array` перегрузку оператора индексации `operator[]`, для того чтобы можно было применять его к экземплярам класса для выполнения обращения к элементам внутреннего массива. Проверить работоспособность оператора

индексации для константных объектов, а также работоспособность при использовании слева от оператора присваивания .

### 3 Результаты работы

3.1 Для сортировки массива была выбрана сортировка слиянием(merge sort) - алгоритм сортировки, который упорядочивает списки или другие структуры данных в определённом порядке. Сортировка разделяется на 3 этапа: Сортируемый массив разбивается на две части примерно одинакового размера; Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом; Два упорядоченных массива половинного размера соединяются в один. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным) или длины 2, тогда, если второй элемент меньше первого, то он меняется с первым местами. Слияние двух подмассивов в третий результирующий массив: на каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1. Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив. Работа программы представлена на Рисунке 1. Алгоритм работы функции сортировки представлен на Рисунке 2. Полный текст программы представлен в Приложении А.

```
./Zadanie1
Файл  Правка  Вкладки  Справка
74 56 81 19 63 9 237 34 7 47
7 9 19 34 47 56 63 74 81 237

56.3554 78.3454 2.82222 19.3425 65 9.34522 456.894 56.7892 8.4 47.45
2.82222 8.4 9.34522 19.3425 47.45 56.3554 56.7892 65 78.3454 456.894

344.097 65.76 66.345 8.645 7.6 344.658 45.876 1.2345 78.353 98.04 44.44
1.2345 7.6 8.645 44.44 45.876 65.76 66.345 78.353 98.04 344.097 344.658

w z u g f e d c b a
a b c d e f g u w z

Press ENTER to continue...
█
```

Рисунок 1 - Работа программы сортировки слиянием

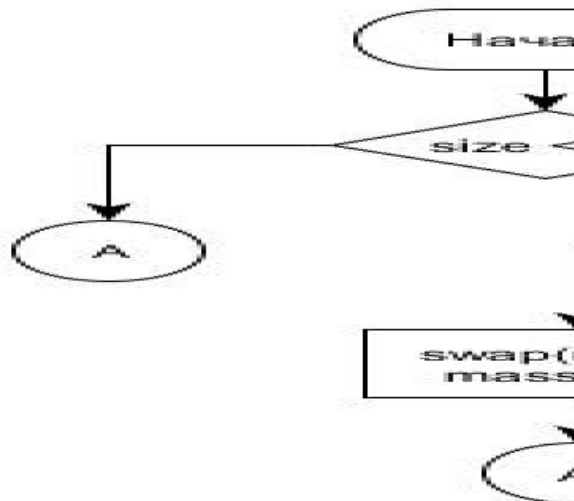


Рисунок 2 - Блок-схема 1

3.2 Работа программы представлена на Рисунке 3. Полный текст программы представлен в Приложении Б.

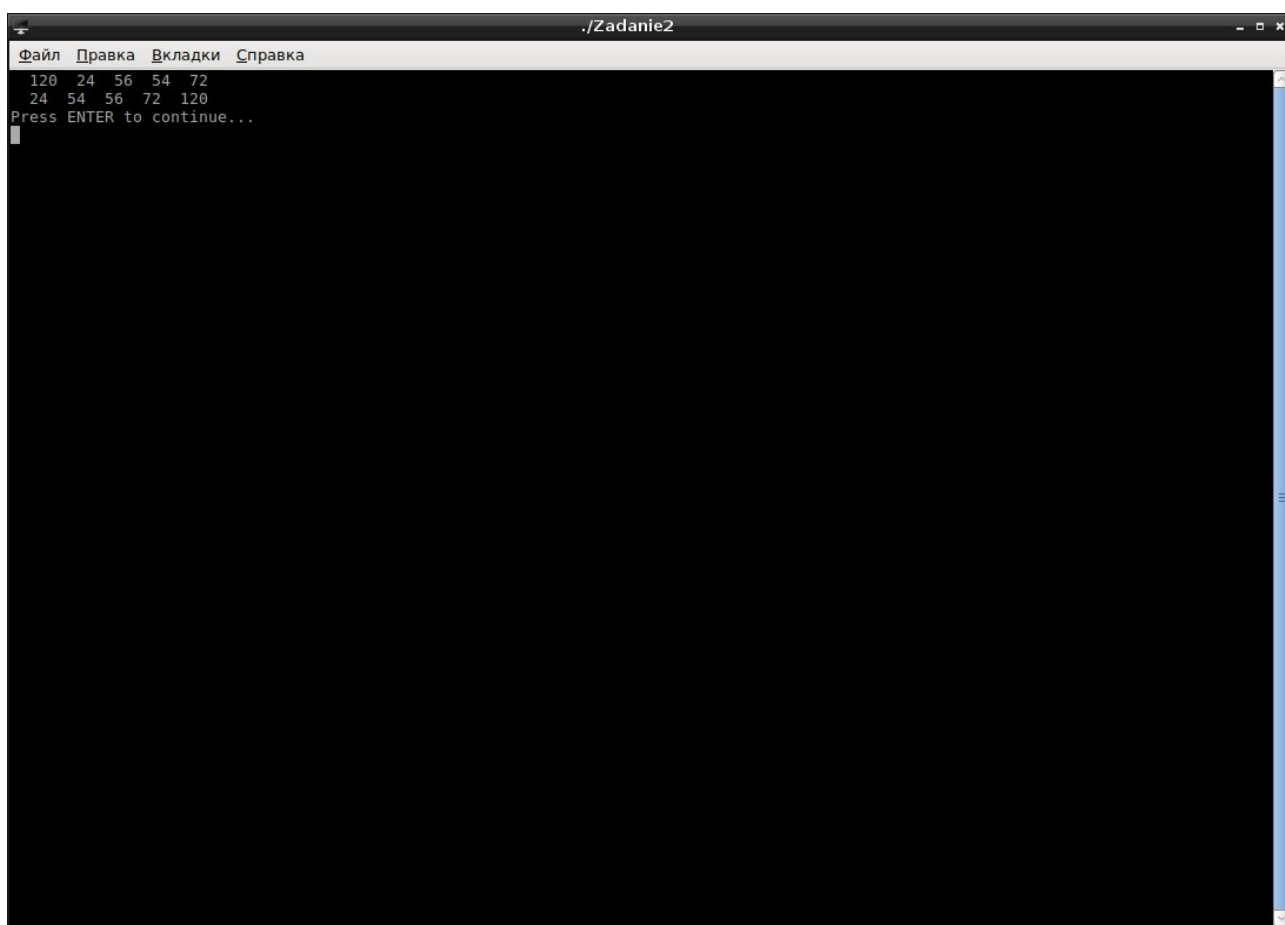


Рисунок 3 - Работа программы класса Rectangle

3.3 Работа программы представлена на Рисунке 4. Полный текст программы представлен в Приложении В.

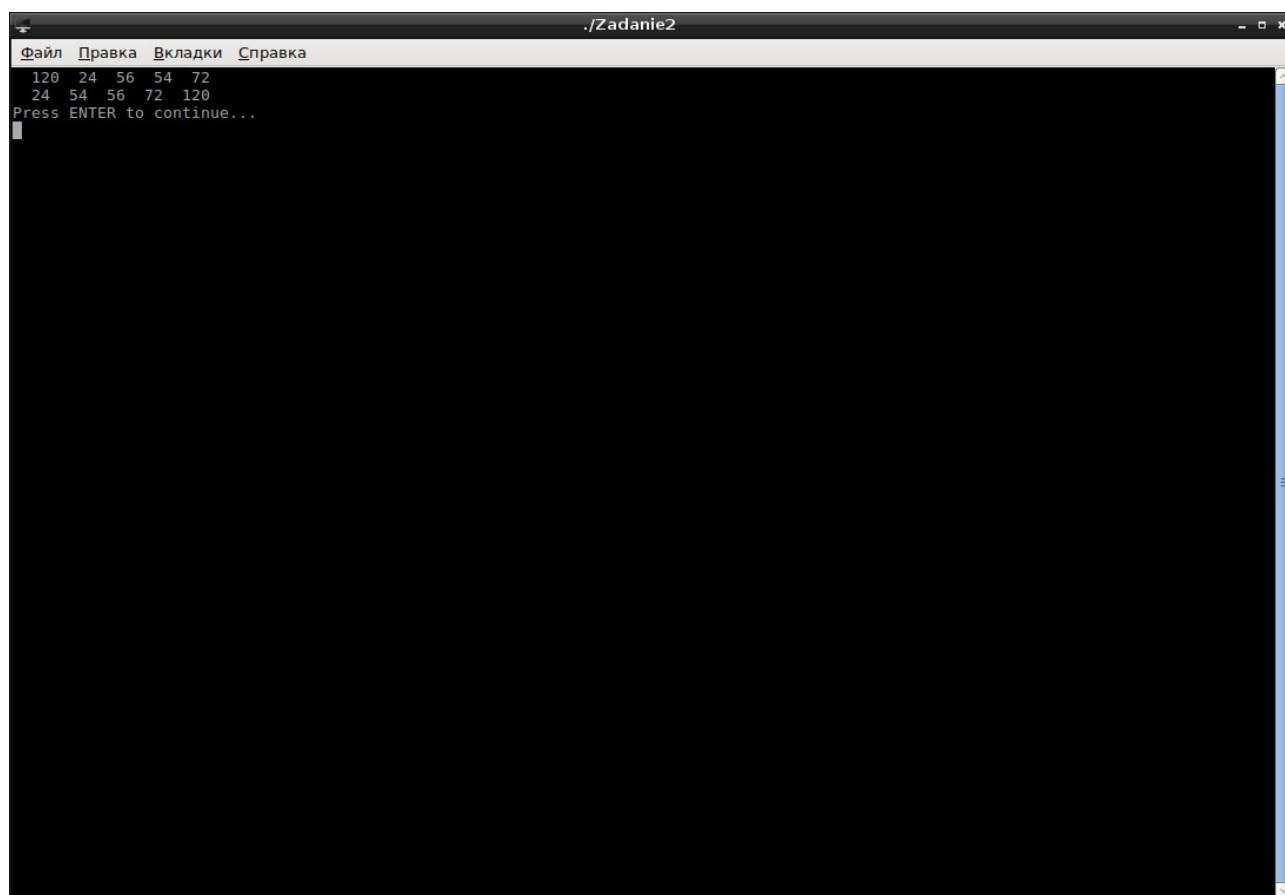
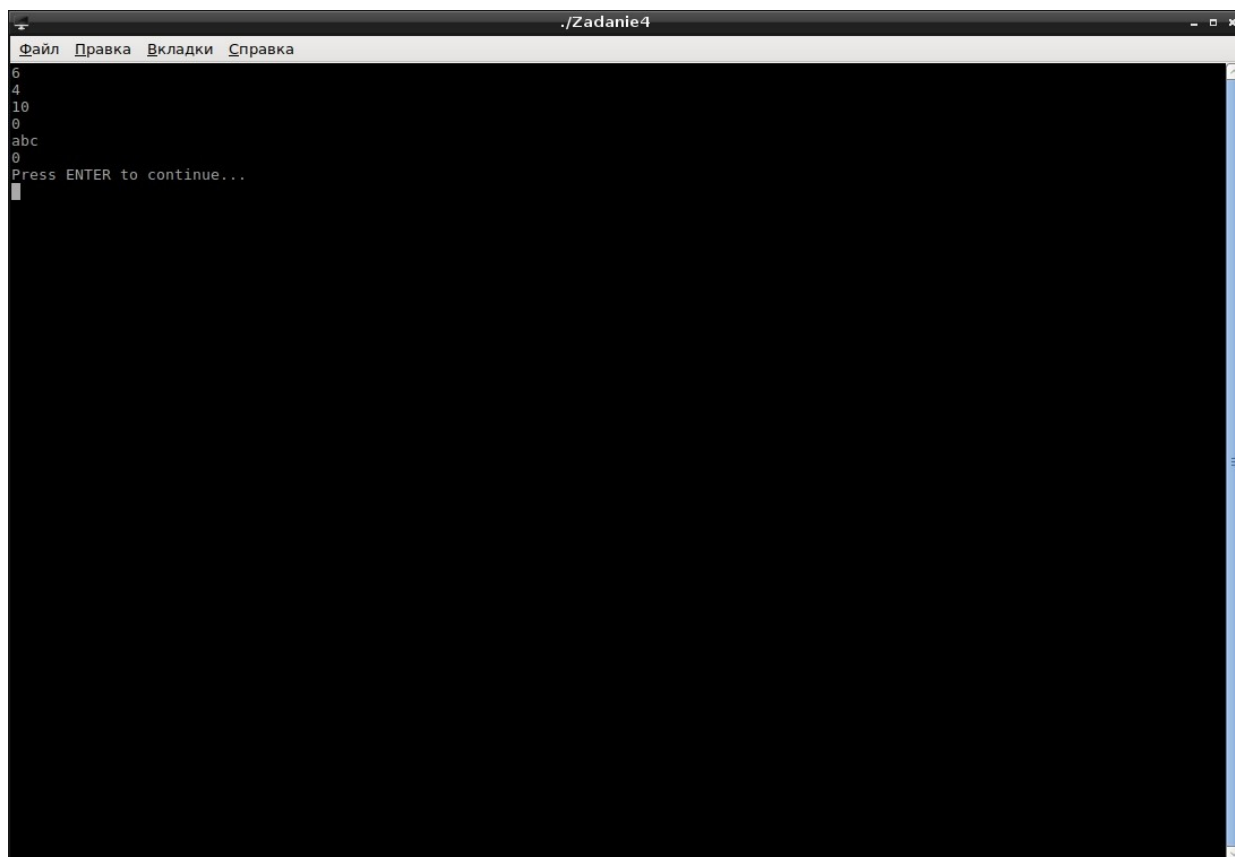


Рисунок 4 - Работа программы класса DoubleBox

3.4 Работа программы представлена на Рисунке 5. Полный текст программы представлен в Приложении Г.



```
6
4
10
0
abc
0
Press ENTER to continue...
█
```

Рисунок 5 - Работа программы класса Array

#### 4 Вывод

В результате выполнения работы были изучены шаблоны языка Си++, а также были написаны шаблонные функции для работы с программами и классами, и получены практические навыки в написании шаблонов классов на с++.



## Приложение А

### Текст программы сортировки слиянием

```
#include <iostream>
#include <iomanip>
#define N 10
using namespace std;

template <typename T> void massivprint(T *mass, int b);
template <typename T> void mergesort(T *mass, unsigned size);
template <typename T> void swap(T *pa, T *pb);
template <typename T> void mergelist(T *mass, unsigned length1,
unsigned length2);
template <typename T> void rightshiftdcycle(T *mass, unsigned left,
unsigned right);
template <typename T> void rightshift(T *mass, unsigned left,
unsigned right);

int main(int argc, char **argv)
{
    // int массив
    int massiv[N] = { 74, 56, 81, 19, 63, 9, 237, 34, 7, 47 };
    massivprint(massiv, N);
    mergesort(massiv, N);
    massivprint(massiv, N);
    cout << endl;
    // double массив
    double massiv1[N] = { 56.3554, 78.345354, 2.82222, 19.3425,
65.0, 9.34522, 456.89422, 56.78922, 8.4, 47.45 };
    massivprint(massiv1, N);
    mergesort(massiv1, N);
    massivprint(massiv1, N);
    cout << endl;
    // float массив
```

```

    float massiv2[11] = {344.0967, 65.76, 66.345, 8.645, 7.6,
344.658, 45.876, 1.2345, 78.353, 98.04, 44.44};
    massivprint(massiv2, 11);
    mergesort(massiv2, 11);
    massivprint(massiv2, 11);
    cout << endl;
    // char массив
    char massiv3[N] = {'w','z', 'u', 'g', 'f', 'e', 'd', 'c',
'b', 'a'};
    massivprint(massiv3, N);
    mergesort(massiv3, N);
    massivprint(massiv3, N);
    cout << endl;
    return 0;
}

```

```

template <typename T> void massivprint(T *mass, int b)
{
    for (int i = 0; i < b; i++)
    {
        cout << "  " << mass[i];
    }
    cout << endl;
}

```

```

template <typename T> void swap(T *pa, T *pb)
{
    T temp = *pa;
    *pa = *pb;
    *pb = temp;
}

```

```

template <typename T> void mergesort(T *mass, unsigned size)

```

```

{
    if (size <= 1)
        return;
    else if (size == 2)
    {
        if (mass[0] > mass[1])
        {
            swap(mass, mass + 1);
            return;
        }
    }
    else
    {
        int length1 = (size - 1) / 2 + 1;
        int length2 = size - length1;
        mergesort(mass, length1);
        mergesort(mass + length1, length2);
        mergelist(mass, length1, length2);
    }
}

template <typename T> void mergelist(T *mass, unsigned length1,
unsigned length2)
{
    unsigned i1 = 0, i2 = length1;
    while (i1 < i2 && i2 < (length1 + length2))
    {
        if (mass[i1] <= mass[i2])
            i1++;
        else
        {
            rightshiftdcycle(mass, i1, i2);
            i1++;
        }
    }
}

```

```

        i2++;
    }
}

template <typename T> void rightshiftdcycle(T *mass, unsigned left,
unsigned right)
{
    T d = mass[right];
    rightshift(mass, left, right);
    mass[left] = d;
}

template <typename T> void rightshift(T *mass, unsigned left,
unsigned right)
{
    for (unsigned i = right; i > left; i--)
        mass[i] = mass[i - 1];
}

```

## Приложение Б

### Текст программы класса Rectangle

```
#include <iostream>
#include <iomanip>
#define N 5
using namespace std;

template <typename T> void massivprint(T *mass, int b);
template <typename T> void mergesort(T *mass, unsigned size);
template <typename T> void swap(T *pa, T *pb);
template <typename T> void mergelist(T *mass, unsigned length1,
unsigned length2);
template <typename T> void rightshiftcycle(T *mass, unsigned left,
unsigned right);
template <typename T> void rightshift(T *mass, unsigned left,
unsigned right);

class Rectangle {
    int h, w, s;
public:
    Rectangle() : h(0), w(0), s(0) {};
    Rectangle(int a, int b)
    {
        h = a;
        w = b;
        s = h * w;
    }
    friend ostream& operator<<(ostream& outputStream, const
Rectangle a);
    bool operator>(const Rectangle s);
    bool operator<(const Rectangle s);
    bool operator==(const Rectangle s);
    bool operator>=(const Rectangle s);
```

```

    bool operator<=(const Rectangle s);
};

ostream& operator<<(ostream& outputStream, const Rectangle a)
{
    return outputStream << a.s;
}

bool Rectangle::operator>(const Rectangle s)
{
    if (this->s > s.s)
        return 1;
    return 0;
}

bool Rectangle::operator<(const Rectangle s)
{
    if (this->s < s.s)
        return 1;
    return 0;
}

bool Rectangle::operator==(const Rectangle s)
{
    if (this->s == s.s)
        return 1;
    return 0;
}

bool Rectangle::operator>=(const Rectangle s)
{
    if (this->s >= s.s)
        return 1;
}

```

```

        return 0;
    }

bool Rectangle::operator<=(const Rectangle s) // Оба оператора +
работают, так в чем же отличие, и какой лучше использовать???
{
    if (this->s <= s.s)
        return 1;
    return 0;
}

int main(int argc, char **argv)
{
    Rectangle b[N] = { {10,12}, {4, 6}, {7, 8}, {6, 9}, {8,
9}, };
    massivprint(b, N);
    mergesort(b, N);
    massivprint(b, N);
    return 0;
}

template <typename T> void massivprint(T *mass, int b)
{
    for (int i = 0; i < b; i++)
    {
        cout << " " << mass[i];
    }
    cout << endl;
}

template <typename T> void swap(T *pa, T *pb)
{
    T temp = *pa;

```

```

    *pa = *pb;
    *pb = temp;
}

template <typename T> void mergesort(T *mass, unsigned size)
{
    if (size <= 1)
        return;
    else if (size == 2)
    {
        if (mass[0] > mass[1])
        {
            swap(mass, mass + 1);
            return;
        }
    }
    else
    {
        int length1 = (size - 1) / 2 + 1;
        int length2 = size - length1;
        mergesort(mass, length1);
        mergesort(mass + length1, length2);
        mergelist(mass, length1, length2);
    }
}

```

```

template <typename T> void mergelist(T *mass, unsigned length1,
unsigned length2)
{
    unsigned i1 = 0, i2 = length1;
    while (i1 < i2 && i2 < (length1 + length2))
    {
        if (mass[i1] <= mass[i2])

```



```

        i1++;
    else
    {
        rightshiftdcycle(mass, i1, i2);
        i1++;
        i2++;
    }
}
}

```

```

template <typename T> void rightshiftdcycle(T *mass, unsigned left,
unsigned right)
{
    T d = mass[right];
    rightshift(mass, left, right);
    mass[left] = d;
}

```

```

template <typename T> void rightshift(T *mass, unsigned left,
unsigned right)
{
    for (unsigned i = right; i > left; i--)
        mass[i] = mass[i - 1];
}

```

## Приложение В

### Текст программы класса DoubleBox

```
#include <iostream>
using namespace std;

template <typename T1, typename T2> class DoubleBox
{
    T1 data1;
    T2 data2;
public:
    DoubleBox(): data1(0), data2(0){};
    DoubleBox(const T1 value1, const T2 value2):data1(value1),
data2(value2) {};
    T1 get1() const;
    T2 get2() const;
    void get() const;
    void set(const T1 value1, const T2 value2);
};

template <typename T1, typename T2> void DoubleBox<T1, T2>::get()
const
{
    cout << data1 << endl << data2 << endl;
}

template <typename T1, typename T2> T1 DoubleBox<T1, T2>::get1()
const
{
    return data1;
}

template <typename T1, typename T2> T2 DoubleBox<T1, T2>::get2()
const
```

```

{
    return data2;
}

template <typename T1,    typename T2>    void    DoubleBox<T1,
T2>::set(const T1 value1, const T2 value2)
{
    data1 = value1;
    data2 = value2;
}

int main(int argc, char **argv)
{
    DoubleBox<int, double> a(5,6.2342);
    cout << a.get1() << endl;
    cout << a.get2() << endl;
    a.set(10, 12.23423);
    cout << a.get1() << endl;
    cout << a.get2() << endl;
    a.get();
    return 0;
}

```

## Приложение Г

### Текст программы класса Array

```
include <iostream>
#include <string>
using namespace std;

template <typename T1, int ArrayLenght> class Array
{
    T1 massiv[ArrayLenght] = {}; // как бы и есть конструктор по
    умолчанию
public:
    Array() {}
    Array(const T1 ini)
    {
        for (int i = 0; i < ArrayLenght; i++)
            massiv[i] = ini;
    }
    Array(const T1 *mass, int size)
    {
        if (size == ArrayLenght)
        {
            for (int i = 0; i < size; i++)
                massiv[i] = mass[i];
        }
        else
            cout << "Ошибка!!! Массивы не одного размера" << endl;
    }
    T1 &operator[] (int index);
    T1 operator[] (int index) const;
};

template <typename T1, int ArrayLenght> T1 &Array<T1,
ArrayLenght>::operator[] (int index)
```

```

{
    if (index < 0 || index > ArrayLenght)
    {
        cout << "Ошибка! Выход за границу массива" << endl;
        exit(1);
    }
    return massiv[index];
}

template <typename T1, int ArrayLenght> T1 Array<T1,
ArrayLenght>::operator[] (int index) const
{
    if (index < 0 || index > ArrayLenght)
    {
        cout << "Ошибка! Выход за границу массива" << endl;
        exit(1);
    }
    return massiv[index];
}

int main(int argc, char **argv)
{
    int mass[5] = { 1, 2, 3, 4, 5 };
    Array<int, 5> b(mass, 5);
    Array<int, 5> a(6);
    const Array<int, 3> c;
    cout << a[1] << endl;
    cout << b[3] << endl;
    b[3] = 10;
    cout << b[3] << endl;
    b[3] = c[1];
    cout << b[3] << endl;
    Array<string, 3> d("abc");

```

```
    cout << d[2] << endl;  
    cout << c[2] << endl;  
    return 0;  
}
```